EXPRESSJS II

Making Express MVC

So far, you've used Express and have done things rather procedurally. This is not the best practice.

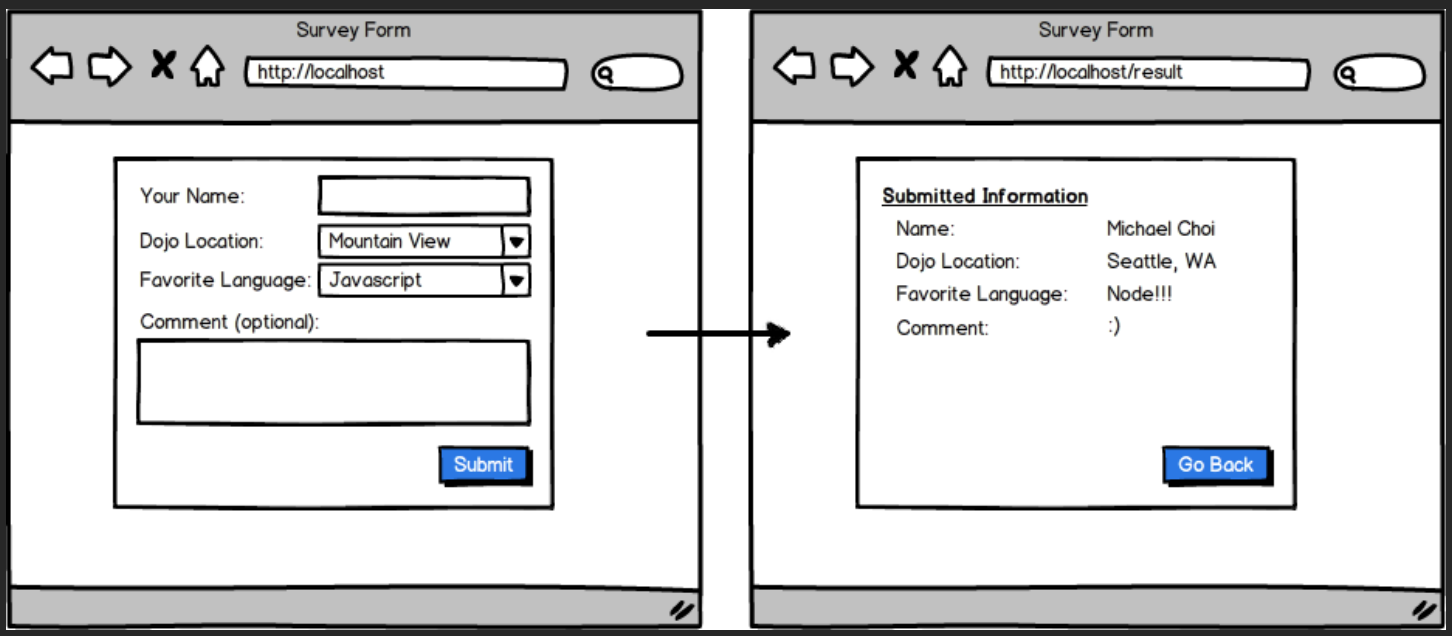
This time, we want you to create your own MVC framework from scratch using Express.

Go back to the PHP Track and pay attention to how the folders and files were organized and we want you to figure out how to modularize Express and make it MVC.

Our recommendation is to make your folder structure as follows:

- **models** folder - put all of your model files here
- **views** folder - put all of your view files here
- **controllers** folder - put all of your controller files here
- **assets** folder - have it host all of your scripts/stylesheets here
- app.js - have this be the main file where starts the server
- config.js - have this store configuration settings for your project including the database credentials
- routes.js - have it contain the routing information

Now, this may seem a bit daunting in the beginning, but do not be afraid. For example, take a look at the previous assignment you did:



Note that most likely, you have a lot of code organized in a single file (e.g. app.js or server.js). If you had to do this using a MVC pattern, you would probably do the following:

1. have app.js load the routes.js
2. have routes.js load the appropriate controllers
3. have the respective controller load the appropriate models
4. have the respective controller load the appropriate view file

For example, you routes.js may look like something as follows:

```
const Express = require("express");
const Router = Express.Router();
const UserController = require(`controllers/users`);
Router.get("/", UserController.index);
Router.get("/login", UserController.viewLoginPage);
Router.get("/register", UserController.viewRegisterPage);
Router.get("/logoff", UserController.processLogoff);
module.exports = Router;
```

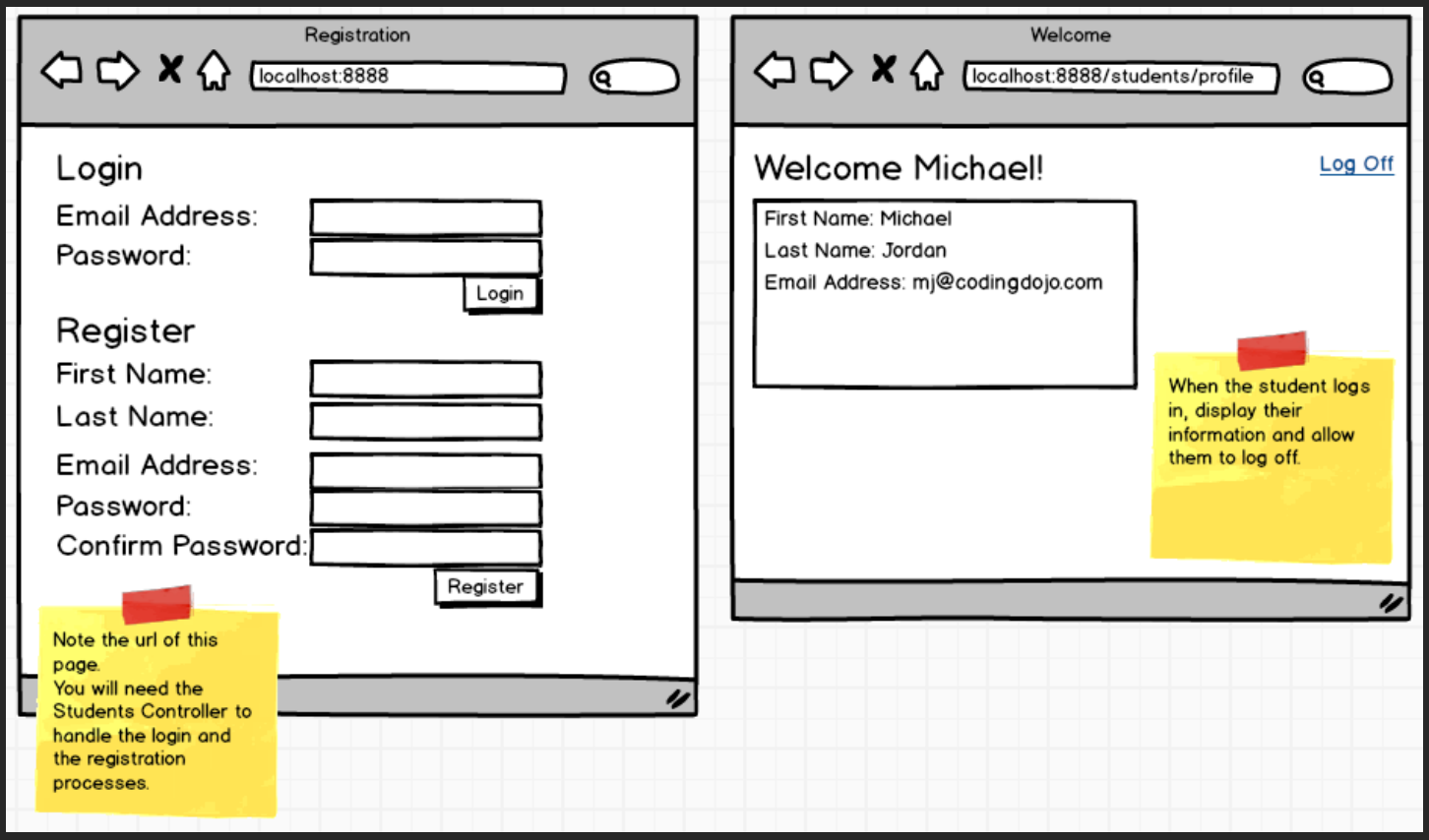
Then in the UserController for example, you would have index, viewLoginPage, viewRegisterPage, or ProcessLogoff page as separate methods within the UserController Class.

As there are lots of learning here, especially as you try to navigate how to modularize your Express app, I won't give you further hints/details but trust that you'll learn how to navigate and create appropriate files and classes. As you do this, I hope you will get more familiar with how to create appropriate files and folder structure.

As this is a difficult assignment (but a key assignment for you to level up your skill as a developer), really try to collaborate with one or two other students to tackle this as a group and to learn from each other.

Login and Registration

Using Express, but also while using the MVC pattern, create from scratch the login and registration, as illustrated below:



Validation Requirements

Make sure you add proper validations so that without a proper email address/password for login, it displays an appropriate error message.

For the registration, also add proper validations so that without a proper first name, last name, email address, or password, the registration fails. Also make sure that if someone already registered with the given email address, it doesn't allow a second registration with the same email address.

Other Security Details

If the user is not logged in, prevent the user from seeing any contents in /students/profile but have it redirect to the main page. This is important for security.

In addition, make sure your code is protected from SQL injection and other common web security vulnerabilities.

Database Requirement

For this particular assignment, build it first using MySQL. You can refer to this documentation to learn how to have MySQL work with Express: https://expressjs.com/en/guide/database-integration.html. Note that you will have to modularize your code so that all the database queries are done through your models. You should have a generic Model class that does common functions (such as connecting to the database, doing a query and retrieving a single result, or

doing a query and retrieving multiple results, etc). Each model should be an extension of that generic Model class that you've created.

Postgres

So far, you've used MySQL for your database needs. Depending on which area you live, the developer community may have strong preference over Postgres vs MySQL. Knowing how to use Postgres is not that difficult and plus, knowing how to use a various database is good for your confidence. It also has the added benefit of making your resume look a lot better.

Now, spend up to 20 minutes to learn about pros/cons of Postgres vs MySQL and then convert your previous Login and Registration assignment (which you have done in MySQL) using Postgres now.

Pay attention to the similarities as well as differences between the two databases. Note that for Postgres, you can search for different GUI available to have you connect and run SQL queries in Postgres database (just like how you used MySQL workbench previously as a GUI for accessing your MySQL database). For example, to see different GUI tools available for accessing a Postgres database, you can read: https://scalegrid.io/blog/which-is-the-best-postgresql-gui-2019-comparison/

Note that the module for Postgres in Express may using traditional callbacks, or promises, or async/await. If possible, use a module that supports either callbacks or async/await (as I personally think promises is not that convenient and makes the code un-necessarily complex).

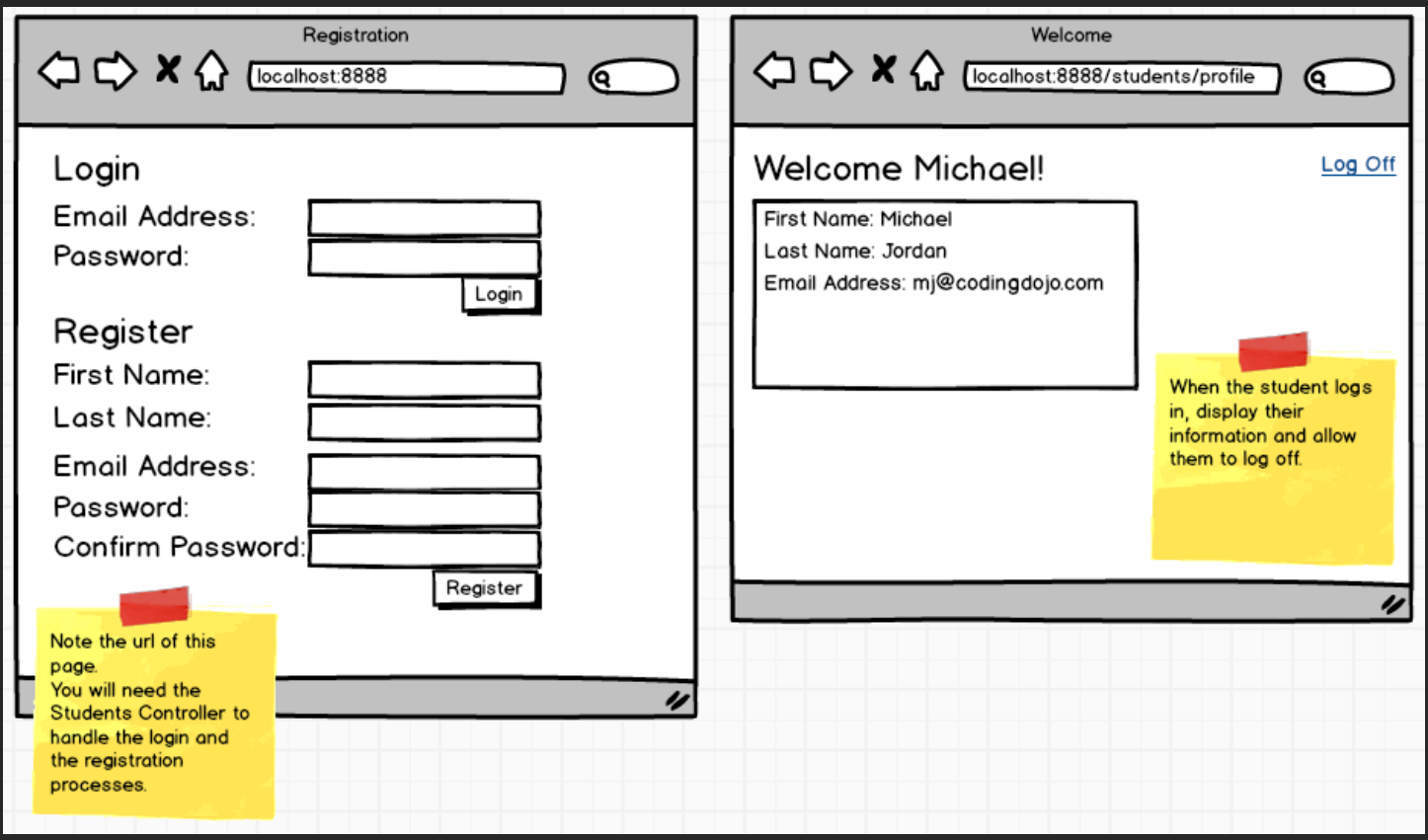
Redis

For Express, by default, sessions are stored in memory and wiped out whenever the server restarts. This means that whenever you work on a feature that requires the user to be logged in, working on that feature/file will trigger *nodemon* to restart the server, which means the session data will be gone, and therefore you need to log into your app again. This could be frustrating and inefficient. To prevent having to login each time the server restarts, you could store your session in a database. Although you could configure your session database to be stored in Postgres, we want you to use a new database called 'Redis' for handling session data.

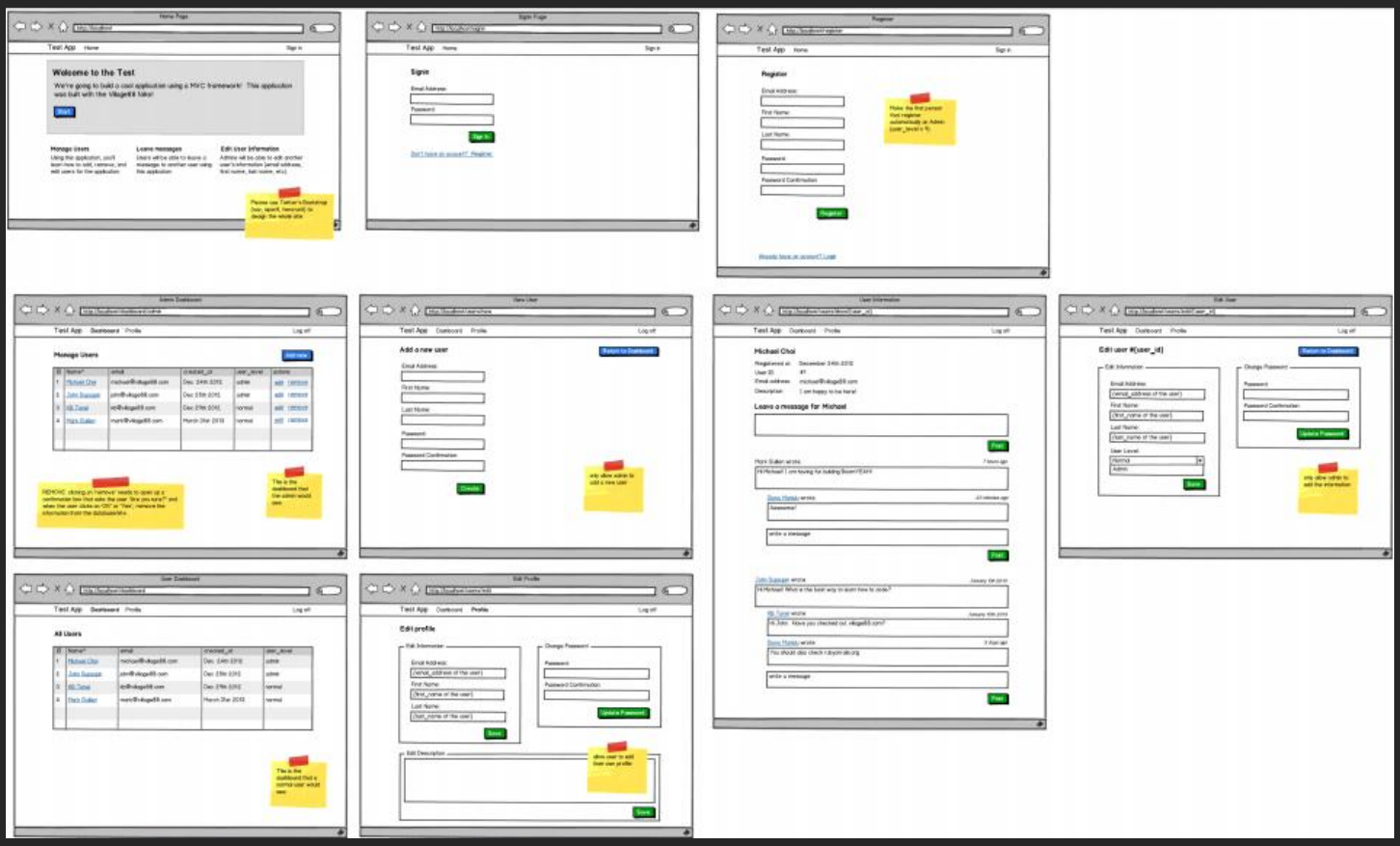
If you're not already familiar with Redis, spend up to 30 minutes to learn about Redis, and then have Redis store your session data. You'll now notice that whenever the server restarts, your session data will remain intact and you won't need to login so frequently.

There are numerous articles/tutorials on how to do this. A good article you can read, that can be a starting point for you, is https://medium.com/mtholla/managing-node-js-express-sessions-with-redis-94cd099d6f2f.

Now that you're familiar with Redis, now change your code so that it utilizes Redis to store the session data. Now, whenever you work on a file that affects what's shown on /students/profile, notice that you won't need to login again!



User Dashboard



Work on the pages indicated in the wireframes and make sure to follow same URL/links structure. For the full PDF, download the handout on the right or click here. Make sure you zoom into the PDF to see a larger view of each page.

EnableProfiler

Remember that in CodeIgniter, there was a profiler that allowed you to see all the database queries that were run to render that page, as well as all the session and post variables? Note that in Express, there is no such profiler.

It's now your turn to create an enableProfiler for your app in Express. The goal is to have you understand how to create these modules from scratch and not be too reliant on other modules that other people have created. Good developers must know how to create their own modules and libraries, and this assignment is designed to have you further understand how to create these modules from scratch and also to boost your confidence that you can create anything you can think of!

Create this enableProfiler that can be used by any controller in your application. Use ES6 class to create this and also look into how you could utilize the concept of 'middleware' to make your code efficient. For example, middleware in Express is discussed here: https://expressjs.com/en/guide/using-middleware.html

(Optional) MongoDB

A lot of Node.js developers like to use MongoDB. MongoDB is a NoSQL database that's commonly used with Javascript/Node.js. If you're ahead of schedule, spend up to 2 hours to learn about MongoDB and up to additional 10 hours to re-factor one of your assignments but where it's now using MongoDB as a database.

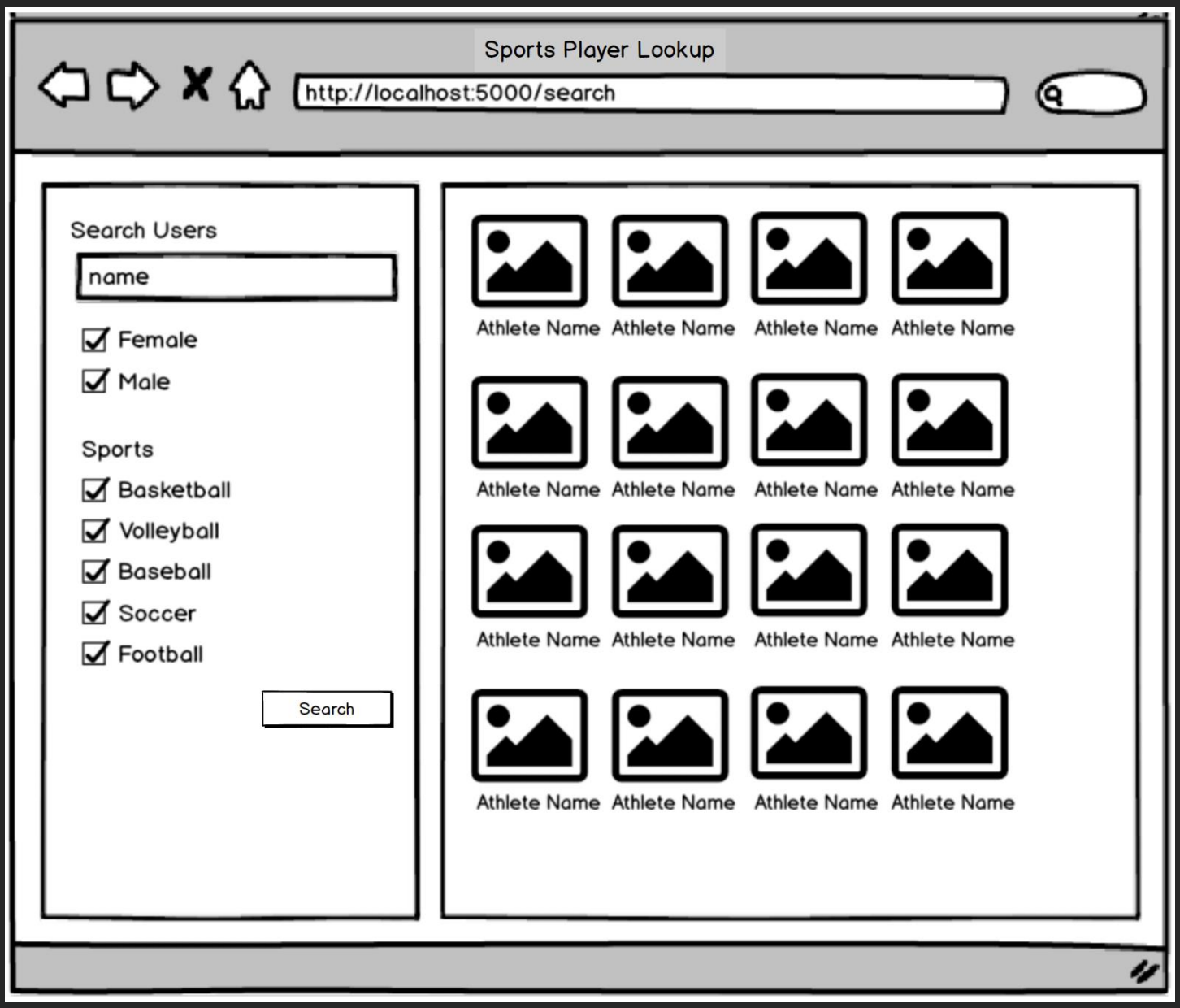
START WORKING ON THIS

Sports Players

Build an application that allows the user to search through a database of sports players where they can filter the results by the name, gender, or which sports they are involved with.

Create your own ERD and pre-populate the database with some fake players information. No need to store the images of the players in your database. Instead, simply store the url of a remote image (an image hosted in another site) in your database (e.g. http://___/__.jpg).

Once you've populated the database with dozens of players information, make this application work.



As you do this, try to make your model as concise as possible. In addition, document your code well so that it's readable.

The goal of this assignment is to help you understand how Ajax can be used together with your Express app.